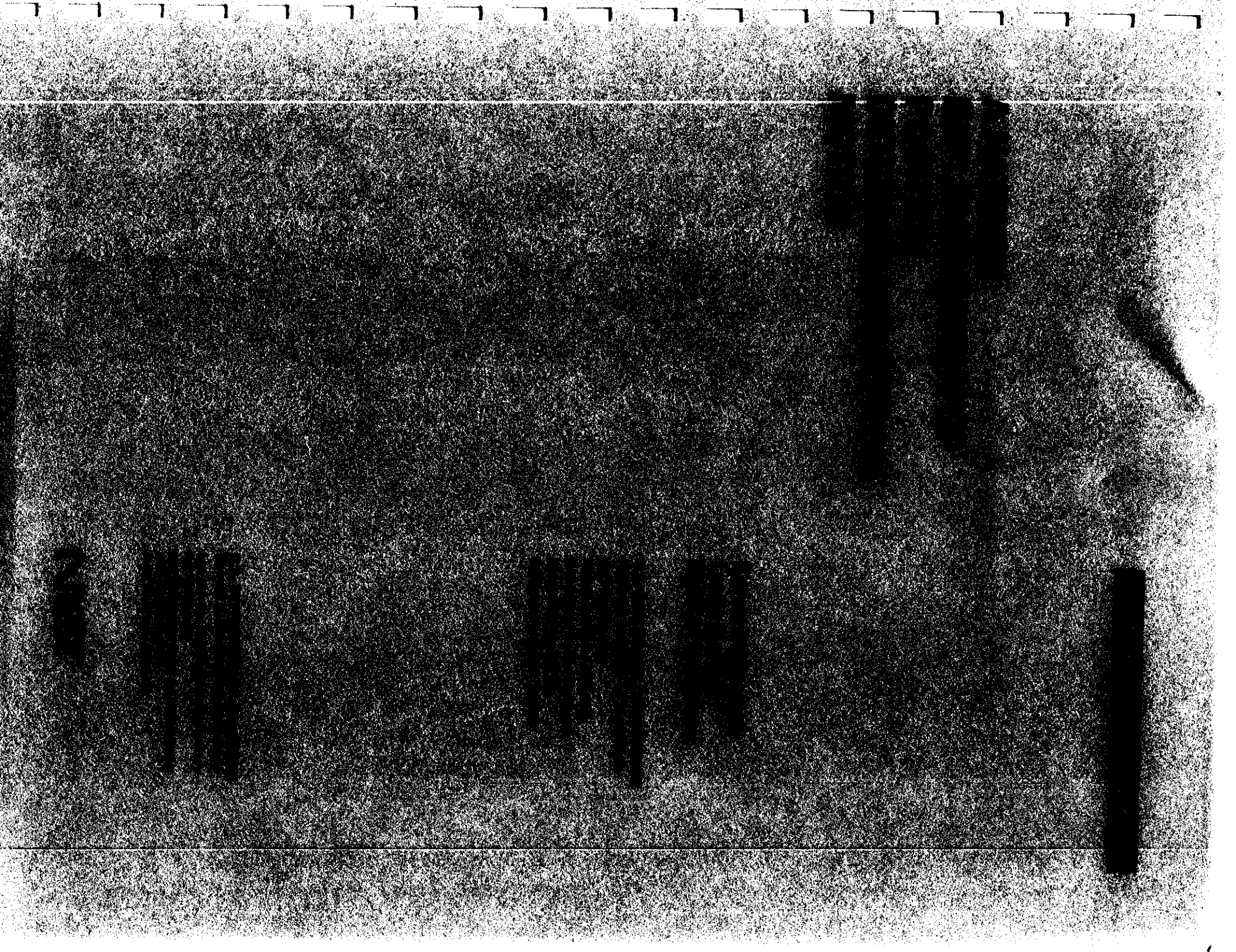
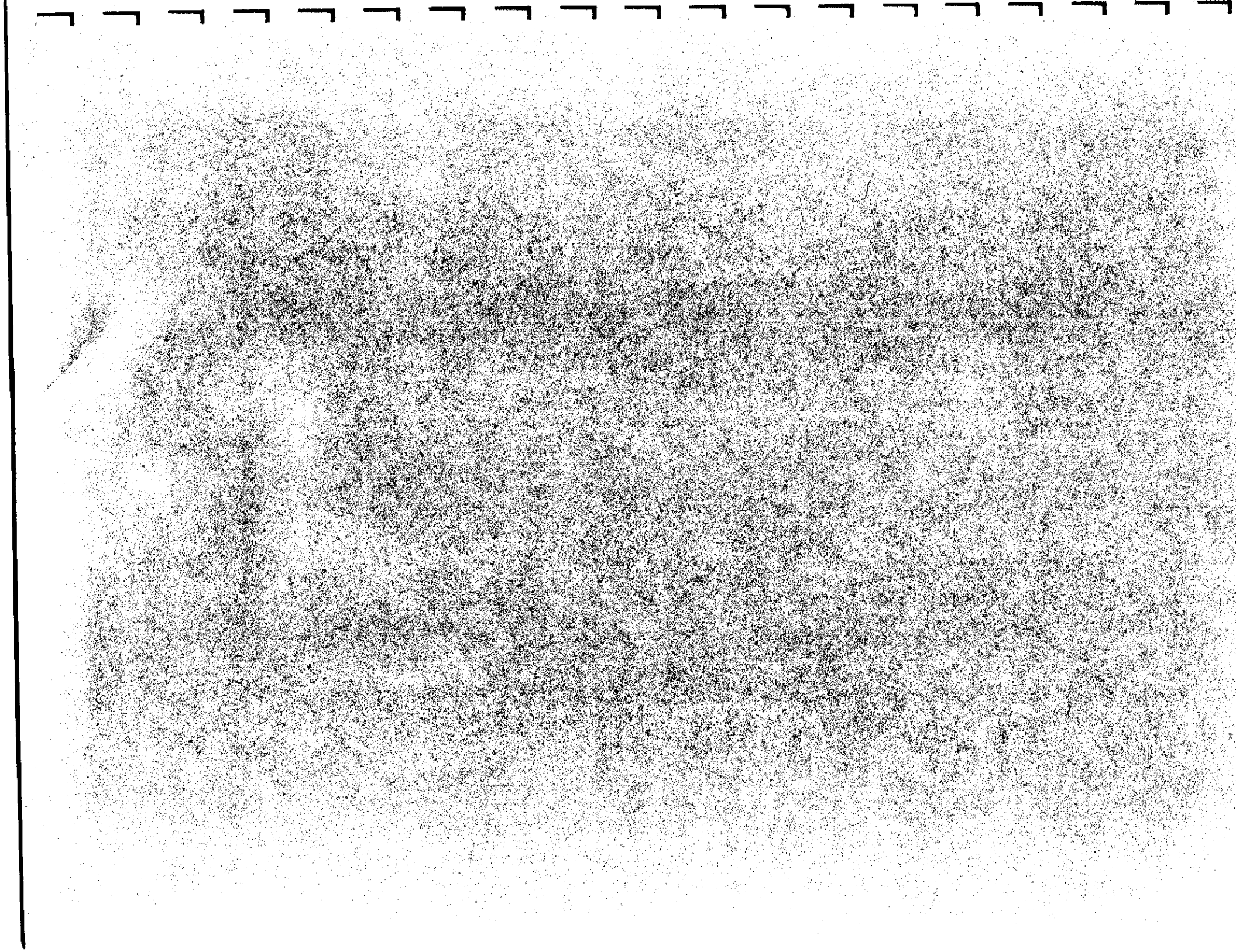




NISTIR 90-4293

A NASREM IMPLEMENTATION OF POSITION DETERMINATION FROM MOTION

Karen Chaconas
Laura Kelmar
Marilyn Nashman

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Robot Systems Division
Intelligent Controls Group
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

April 1990

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
Lee Mercer, Deputy Under Secretary for Technology
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

A NASREM IMPLEMENTATION OF POSITION DETERMINATION FROM MOTION

Intelligent Controls Group
Robot Systems Division

Principle Authors:

Karen Chaconas

Laura Kelmar

Marilyn Nashman

Date: January 16, 1990

Document number: ICG-# 24
Document version: 1.0
Document approval:_____

Revisions

1. Revision Author:

   Revision Date:

   Purpose:

Scope of the Document

This document describes a NASREM implementation of a system which determines 3^D position from motion. The system performs real-time image processing to extract the two-dimensional centroid of a moving object. It then employs an inverse perspective algorithm to transform the centroid of the moving object on a planar surface to a three-dimensional position. The position is used by the control system for a robot manipulator which tracks and catches the moving object.

1. Introduction

Real-time trajectory generation based on sensory feedback, such as vision and proximity, allows performance of tasks based on sensed data rather than a priori knowledge. This provides for robustness of task execution despite incomplete or uncertain knowledge of the environment. Robot positioning systems base their corrective motions on the difference between the desired, or reference, joint positions and the actual ones. Because of possible errors in the manipulator kinematic model and/or the locations of object in the workspace, the relative position of the end-effector and the object may be in error. If joint encoders provide the only sensory feedback, then there is no way to determine the final Cartesian positioning error since no direct measurement of the final end-effector position is available. With the addition of perceptual sensory feedback, the control system can more reliably relate the relative pose of the manipulator's end-effector to objects (cues) in the world.

In this paper we discuss an initial task performed in our laboratory involving an implementation of the lowest two levels of the NASREM control hierarchy. Our telerobot system consists of a manipulator, a gripper, and a camera. The processing modules for this subset of the NASREM architecture can be recombined according to their function in the system, as shown in Figure 1. The system consists of two main *branches*; the left branch contains the perception processes and the right branch contains the manipulation processes. The perception processes provide sensory

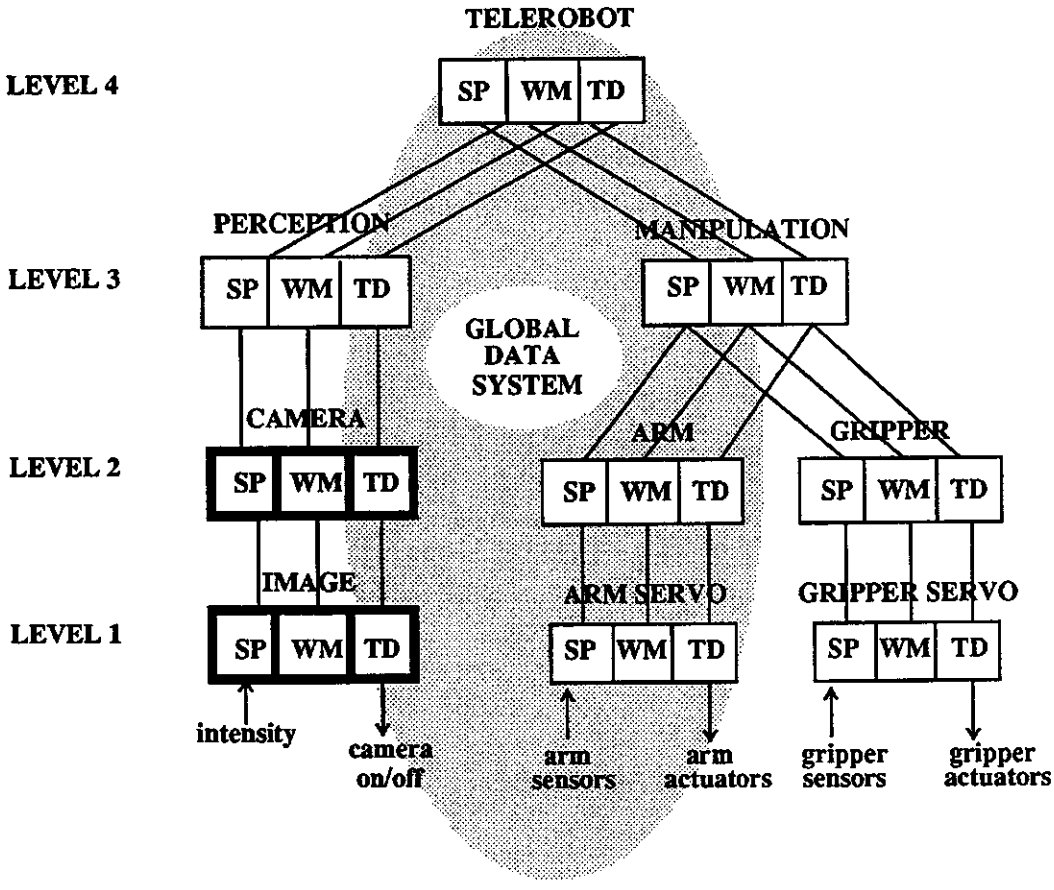


Figure 1. A NASREM Subsystem

feedback from the camera; the manipulator processes plan and execute manipulator trajectories. Note that while the two branches decompose tasks independently within each branch, communication between processes, both within a branch and across branches, occurs via the global data system. The perception and manipulation branches cooperate during trajectory generation and execution. The demonstration successfully integrates the sensory processing, world modeling and task decomposition modules, thus closing the sensory feedback loop.

The discussion in this paper focuses on the implementation of modules highlighted in figure 1. In the following sections we present an overview of the system in our lab. We introduce the role of the operator interface for initializing the world model. We then proceed to describe the algorithms that we implemented, and we characterize the information passed to the modules in NASREM: Levels 1 and 2 of the visual perception branch and a specialized hardware interface module which allows for transfer of information to a specific image processing machine.

2. System Description

In the experiment, a small ball is released at the top of an inclined board which contains randomly placed pegs, as shown in figure 2. As the ball rolls down the board, it is tracked by the single camera vision system and then caught by the manipulator at the bottom of the board [7]. Our manipulator is a Robotics Research[1] seven degree-of-freedom arm. The vision processing is performed by a real-time image processing machine, the Pipelined Image Processing Engine (PIPE), which is commercially available through Aspex, Incorporated. The software is written in

1. Certain commercial equipment, instruments, or materials are identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by NIST, nor does it imply that the materials or equipment identified are necessarily the best for the purpose.

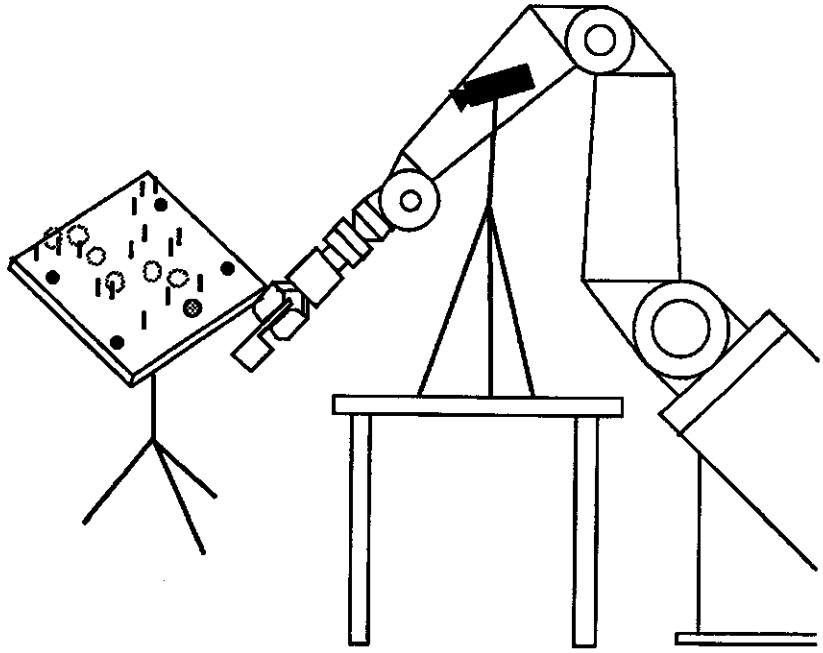


Figure 2. Ball catching task setup.

Ada and runs on 68020 processors.

This paper details the role of the sensory processing and world modeling modules in providing feedback to the manipulation modules. The NASREM implementation is based on the concept of cyclically executing modules which serve as the computational units for the architecture. After initialization, all computations are performed by cyclically executing processes which communicate via global read-write interfaces. Each unit acts as a process which reads inputs, performs computations, and then writes output. Such a process always reads and executes on the most current data; it does not wait for new data to arrive. Thus, reliable cyclic execution requires that a module be able to read or write data with minimal delay. Reading and writing involves the transfer of data between local buffers and buffers in global memory. System software has been written to prevent data corruption during these transfers. The organization and operation of cyclically executing processes is fully described in [6,8].

Since special support hardware is used, specifically PIPE, a separate process exists for interfacing between Level 1 and PIPE. This process, called the PIPE manager, also is a cyclically executing process that is written in Ada and differs from other computational units only in that it writes commands to special purpose hardware. The interface between PIPE and the PIPE manager is a VME interface which provides direct memory access between the PIPE and the target system.

3. Operator Interface for Perception Initialization

The perception processing modules perform initialization of the board's position with respect to the camera. The camera is arbitrarily positioned at the start of the task; the only constraint on its placement is that the entire face of the board must be in the field of view. World modeling then must record the camera's location in the world. The plane of the board is denoted by four black circles with a known configuration. We determine the transformation from the board to the camera using a four coplanar point algorithm [11]. In addition to the camera-to-board transformation, the board's pose in world coordinates must also be known. The position and orientation of the board is identified in world coordinates by placing the manipulator's end effector at three locations on the board and calculating the plane which contains all three points. Now, x, y, z locations produced with respect to the camera's frame of reference can be transformed to the board's reference frame and from there to a world coordinate system.

In our implementation, the operator interface facilitates initialization of the system in the absence of a complete world model. The operator interface provides a means by which human operators can observe, supervise, and directly control the system [1]. The operator interface is used to locate the position of the four circles in the camera image. By moving the keyboard mouse, the operator is able to isolate each of the four black circles within a "window of interest." Once the centroids of the four circles are found, world modeling computes the transformation from the board to the camera.

The operator interface performs the initialization as shown in figure 3. In this figure, the processes that are pertinent for initialization are shown using rectangles. The ovals represent the interfaces between processes and detail the data that is passed. The trapezoids show the pipelined processes that perform image processing. They operate in sequence within a cyclically executing process. In the current implementation, the operator interface is a stand-alone unit which is called before running the perception modules. The operator moves a keyboard mouse which causes a cursor on his screen to move. The operator interface process reads the position of the cursor on the

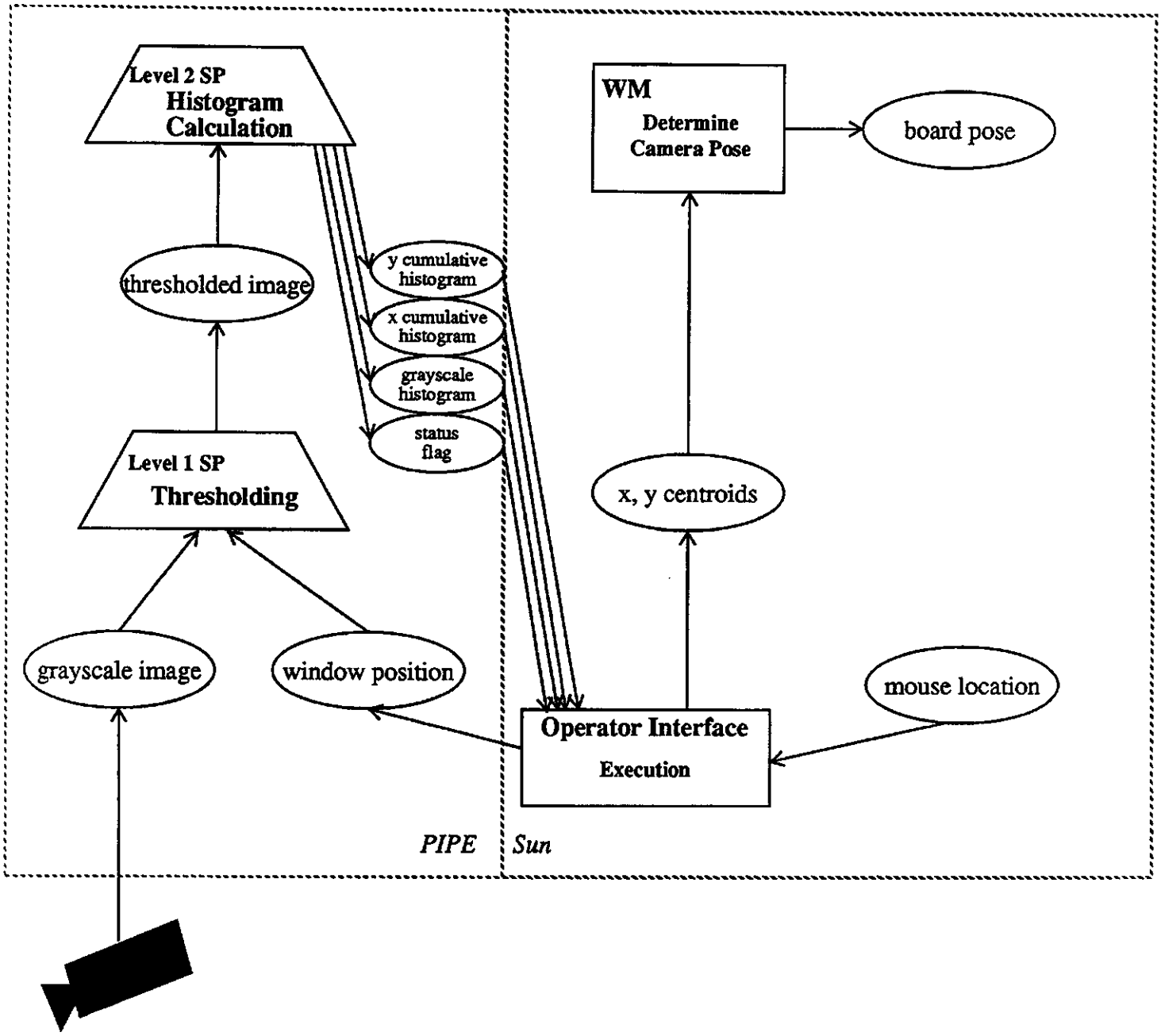


Figure 3. Perception Initialization using Operator Interface.

screen in a scale that corresponds to image coordinates. The mouse position information is sent to PIPE where it is used to determine the location of a mask, or "window of interest", in the image. The mask is centered on the position of one of the four circles in the camera image of the board. The purpose of the mask is to blank out all information other than that in the area of interest.

A grayscale image, *I(i,j)*, from the camera is digitized by PIPE, where a Level 1 sensory processing algorithm is running. The Level 1 sensory processing algorithm produces a thresholded image, *I'(i,j)*, using the function:

$$I'(i,j) = \begin{cases} 0 & \text{when} \quad I(i,j) < t \\ 1 & \text{when} \quad I(i,j) \geq t \end{cases} \qquad (1)$$

for all pixels *i,j* in the image. The threshold value, *t*, can be any value between 0 and 255; in our case, the operator chose beforehand to set *t* to 80.

The resulting thresholded image is processed by a Level 2 sensory processing algorithm on the PIPE to obtain grayscale and cumulative histograms. The grayscale histogram produces the values ΣΣf(x, y) used in equations (2) and (3) by determining the total number of occurrences in the thresholded image of each value from 0 to 255. The x (or y) cumulative histogram produces the values ΣΣxf(x, y) (or ΣΣyf(x, y)) used in equation (2) (or (3)) for calculation of the x (or y) centroid. The cumulative histogram gives the summation of the occurrence of each value from 0 to255 in the binary image multiplied by its x (or y) location.

The operator interface process uses the output of the PIPE algorithm to compute the centroid $(\bar{x}, \bar{y})$ of all pixels above the threshold value. The operator interface process polls the PIPE until a status flag is set by the running algorithm indicating that updated results are available.The process reads the appropriate memory-mapped locations which contain information relating to the histogram and the cumulative histograms. Since the image is binary and only the non-zero values are of interest, only one location is read from each histogram. The centroids are computed according to the equations:

$$\bar{x} = \frac{\sum_x \sum_y x f(x,y)}{\sum_x \sum_y f(x,y)} \qquad (2)$$

$$\bar{y} = \frac{\sum_x \sum_y y f(x,y)}{\sum_x \sum_y f(x,y)} \qquad (3)$$

where $x = \{0..\text{the width of the image in pixels}\}$, $y = \{0..\text{the height of the image in pixels}\}$ and $f(x,y)$ is the thresholded value at the pixel position x, y.

The number of pixels contributing to the centroid computation can change. The intensity value

of a pixel can fluctuate due to sampling noise or variations in the ambient lighting. Any change in measured grayscale of pixels whose intensity is close to the threshold value in equation (1) can cause the resulting value to not remain constant at either 0 or 1. In order to obtain as accurate a centroid calculation as possible, 10 histogram readings are read by the operator interface process. The grayscale histogram reading with the largest area and the corresponding cumulative histograms are used for the centroid calculation.

The centroid of each of the four circles is computed in the same manner and is stored in global memory for use by world modeling. All four centroid positions are used by the Level 2 world model process to compute the camera's pose with respect to the board. Once the pose is known, two dimensional image positions corresponding to points on the board can be transformed to three-dimensional positions which are used by the manipulation processes.

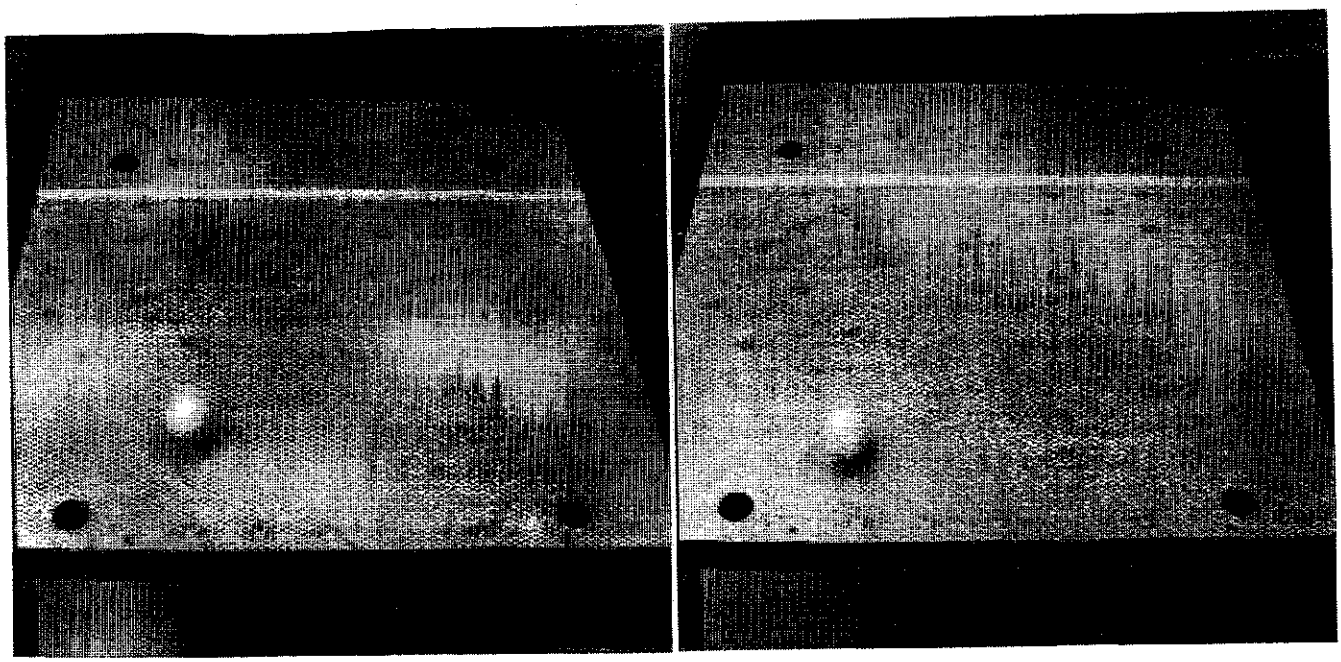
4. Vision Processing

The sensory processing system must extract a feature which can be used to identify the three-dimensional location of the ball at each sampled point in time. The algorithm must produce updated outputs as rapidly as possible in order to provide useful feedback to the world model and the manipulator modules. In addition, it must be able to demonstrate our system in a relatively unstructured laboratory environment. We use standard overhead fluorescent lighting; we do not use special backdrops to separate the camera field of view from the remainder of the laboratory. Since the ball that we use is similar in intensity to the planar surface we do not use a traditional thresholded intensity image since the ball would not be separable from the background in the image. For this reason, we use an algorithm that segments the image based on motion due to changing intensity values between successive images. This approach segments the moving ball from the static background.

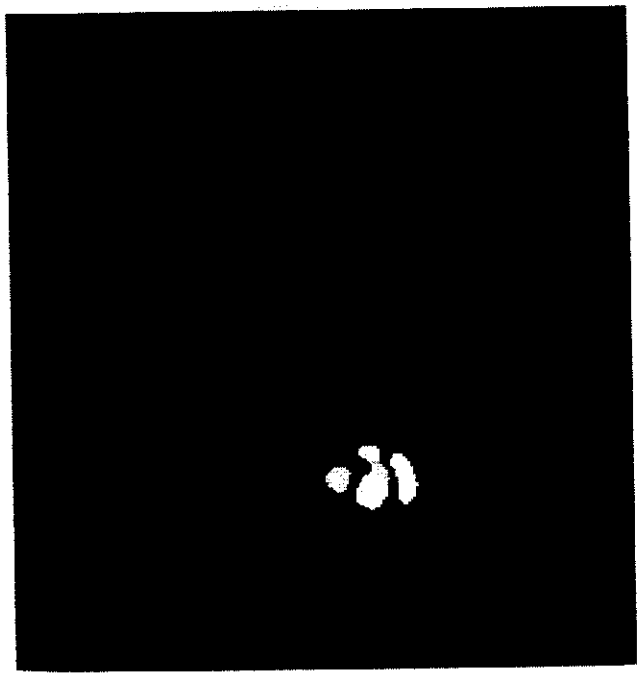
The vision algorithm used for the experiment determines the location of the ball using a difference of images algorithm. PIPE acquires images at a video rate of 1/60 second. However, the input images that are used are taken every 1/30 second to ensure that they have the same camera field sampling. An example of two images taken of the ball in motion and separated by 1/30 second are shown in figures 4(A) and 4(B). Incoming images are smoothed using a Gaussian convolution to diminish the effects of spurious noise in the image [3]. Two consecutive smoothed images are subtracted from each other in order to detect any change in intensity due to motion between the frames. All non-moving features in the field of view "disappear" in this difference image since the grayscale value of a pixel in the second frame is being subtracted from the identical grayscale value in the first frame. The result of the difference image is thresholded to produce a binary image in which white areas refer to moving points and black areas refer to stationary points. This result is shown in figure 4(C). PIPE uses the thresholded image to extract information relating to the centroid of the moving points.

5. Implementation of Software Processes

As mentioned earlier, the algorithms for this demonstration are implemented in Levels 1 and 2 of the perception hierarchy. The modules that support position determination of a moving object are shown in figure 5. The modules that are shown using rectangles are those used to read commands, execute procedures, and write results. The ovals detail the data being passed between processes. The trapezoids show pipelined algorithms that execute together to perform image



(A) (B)



(C)

Figure 4. Intermediate Results of the Sensory Processing Algorithm
(A) Input image at time t=n (B) Input image at time t=n+1/30 (C) Result of motion detection

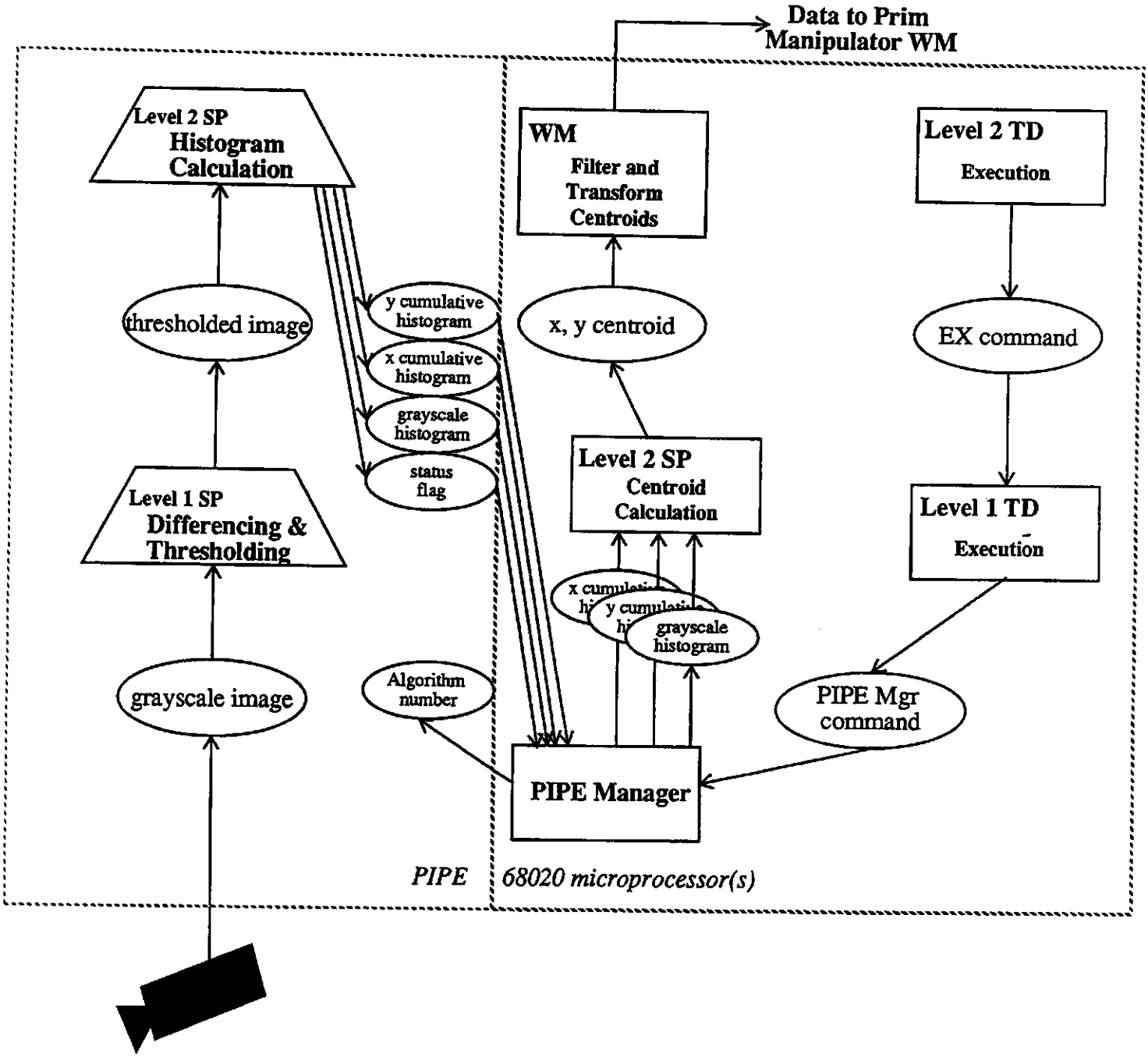


Figure 5. Perception Execution.

processing computations. The command for execution of a sensory processing algorithm to perform centroid extraction of a moving object originates at Level 2. It proceeds through the task decomposition system from Level 2 to Level 1. A command then is sent to the PIPE manager process which initiates the execution of a sensory processing algorithm on the PIPE. The sensory processing algorithm at Level 1 operates on incoming images, and the results are passed to the Level 2 sensory processing modules where the next part of the centroid computation occurs. The results are passed to the world model process for transformation to three-dimensional positions. These steps are described in more detail below.

5.1. Task Decomposition

Each command is processed through the levels of the task decomposition hierarchy. The purpose of the task decomposition system in the perception hierarchy is threefold. It must prioritize incoming commands; select the most appropriate algorithm for the sensory processing system to perform; and choose the parameters to use for a given algorithm. For this demonstration, the architecture has been reduced to a skeletal form while still preserving the NASREM requirements. For example, the Job Assignment (JA) module discussed in [3] is not implemented since there is only one source issuing a command. Thus there is no need to prioritize incoming requests. The Planner (PL) module is not implemented because the decision concerning which algorithm to run was made off-line and is pre-programmed into our system. The functions performed by the Execution (EX) module are not required to select algorithm parameters since the parameters are stored within the algorithm and not modified during execution. The Level 2 EX module initiates a command for the calculation of a two-dimensional centroid and specifies that this command be continuously performed. The command is written to the Level 1 EX module via a reader-writer communication package that is described in [6]. The Level 1 EX module reads the command and issues a command for motion detection to the PIPE manager.

5.2. PIPE Manager

The PIPE manager process acts as an interface between the sensory processing modules and PIPE. It is a cyclically executing process which reads commands written by the sensory processing modules, activates PIPE and monitors its execution, and finally writes status and outputs to global memory buffers.

After reading a command from the task decomposition module, the PIPE manager interprets and activates the appropriate algorithm on PIPE. On the PIPE, a program can contain up to 16 separate algorithms. A system controller specifies which algorithm is executing at any given time. The PIPE manager converts the requested algorithm name to the corresponding PIPE algorithm number. The PIPE algorithm number is sent to the PIPE system controller to initiate execution of a specific algorithm residing on the PIPE.

The PIPE manager then polls the PIPE until a status flag is set by the running algorithm indicating that updated results are available.The process reads the appropriate memory-mapped locations which contain information relating to the histogram and the cumulative histograms. The histogram produces the values $\Sigma\Sigma f(x, y)$ used in equations (2) and (3) by determining the total number of occurrences in the motion image of each value from 0 to 255. Since the difference image is binary and only the non-zero values indicate motion, only one location is of interest in the histogram. The cumulative histogram produces the values $\Sigma\Sigma xf(x, y)$ (or $\Sigma\Sigma yf(x, y)$) used in

equations (2) and (3) for calculation of the x (or y) centroid. The cumulative histogram gives the summation of the occurrence of each value from 0 to255 in the motion image multiplied by its x (or y) location. Again, since the motion image is binary only one location in each of the cumulative histograms is of interest. All three of the values from the histogram and the cumulative x and y histograms are written to the Level 2 sensory processing module via a reader-writer communication.

5.3. Sensory Processing

The PIPE algorithm performs both the Level 1 and Level 2 sensory processing functions. In this case, the selected algorithm performs the Level 1 task by first subtracting two images and then thresholding the resulting image. The output is an image segmented on the basis of motion. The next part of the algorithm, executed on the PIPE's iconic to symbolic mapper, performs the Level 2 task. It uses the binary motion image to compute the gray level histogram and the x-direction and y-direction cumulative histograms. These values represent a count of the total number of image points which are in the object and the sum of the x and y pixel coordinates of those points. The Level 2 task reads the histogram values written by the PIPE manager. It unpacks this data from the PIPE format and uses the results to compute the two dimensional centroid according to equations (2) and (3). The results of the computation are written to the world model support module via a reader-writer communication.

5.4. World Modeling

World modeling performs several functions in support of the calculation of a three-dimensional position [8]. The image differencing algorithm described above is useful for tracking a ball moving on a planar surface when the ball is the only object which moves in the field of view. In that case, the centroid successfully reflects the location of the ball. However, if the integrity of the scene cannot be guaranteed, world modeling must intervene. One method of isolating the motion of the ball is for world modeling to provide a window of interest around the expected location of the ball in each image. The window locations would be based upon previous reading(s) and knowledge about the movement of the ball between sampling instants. An alternate method, the one which we employ currently, is to maintain a history of centroids which world modeling uses to filter out spurious readings. In our current implementation, world modeling performs all filtering after transforming the image centroid to3^D space.

The centroid value, as supplied by sensory processing, is a 2^D centroid in image space; the manipulator Primitive module (Prim) requires a 3^D position with respect to a given reference frame. World modeling performs several computations to transform the 2^D centroid data to a value usable by Prim. First the location of the ball's centroid in the image is represented as a 3^D point on the camera's image plane. The ball's position on the inclined board is computed by projecting a ray from the camera's optical center through the point on the image plane. The intersection of the ray with the inclined board represents the ball's location in 3^D space.

Specifically, first the point representing the origin of camera's optical axis and the point giving the ball's position on the image plane are represented with respect to the board's frame of reference. They are transformed to the board's frame of reference via the transform computed during initialization using the four coplanar point algorithm. Then the equation of the line containing the two points is given by:

$$\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1} = \frac{z-z_1}{z_2-z_1} \qquad (4)$$

where (x_1, y_1, z_1) is the ball position and (x_2, y_2, z_2) is the camera position. The equation of the plane of the board can be represented by the general plane equation:

$$\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1} = \frac{z-z_1}{z_2-z_1} = t \qquad (5)$$

where the variable t parameterizes the plane. The line through the two points (the origin of the camera and the ball's point on the image plane) intersects the plane of the board at z=0. Actually, a more precise value for z at the point of intersection is (z = half the diameter of the ball), not (z=0). The non-zero value for z reflects the volume, or depth, of the ball; the value generated by the sensory processing algorithm, representing the centroid of the moving ball, is affected by the ball's depth. Using the known z location of the ball, we solve equation (5) for t and then determine the x, y location of the ball.

After determining the 3^D location of the ball, world modeling filters out spurious or invalid data points corresponding to locations beyond the boundaries of the inclined board [9]. World modeling also filters out locations which are displaced from the previous reading by more than a predefined limit. We set the limit roughly based upon the physical setup (e.g., incline angle of the board) and the time delay between samples. World modeling transforms the point to the world coordinate system using the transform between the board and the base of the robot manipulator ($^{board}T_{world}$) that is stored during initialization. The actual position of the ball, as computed from the image, is of little use to Prim. In order to catch the ball, the manipulator must move toward predicted locations of the ball. The manipulator Prim algorithm for generating manipulator goal states based upon the ball position is discussed in [7].

6. Timing

The distribution of each of the software processes on microprocessor boards is based on timing requirements. In order to determine the optimal update rate, we must look at the Primitive (Prim) module for the manipulator. Three-dimensional ball positions must be updated as rapidly as possible since the Prim module uses a 5 ms update rate for sending commands to the Servo level.

It is evident that the image processing to perform motion detection and centroid calculation needs to be done as rapidly as possible. The NIST implementation uses PIPE, specialized image processing hardware designed to operate on 64K bytes of image data every 1/60 second. There are many ways to program PIPE to perform motion detection and centroid computations. The following examples illustrate two such implementations and their effect on throughput rate (the time interval between digitizing an image and outputting results) and update rate (the time interval

between successive outputs).

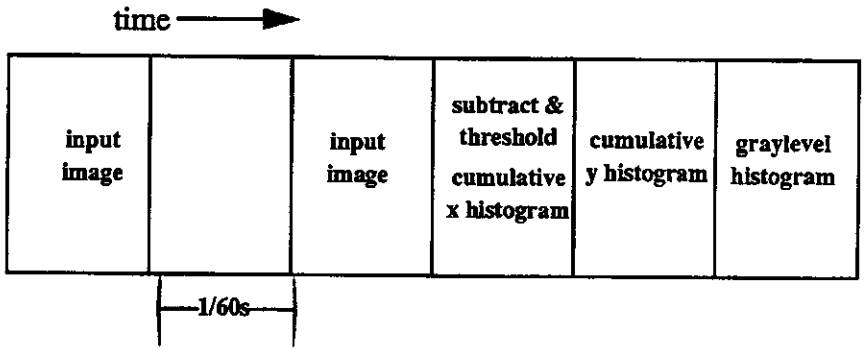


Figure 6. PIPE Implementation Optimized for Throughput Rate

The implementation shown in figure 6 maximizes the efficiency of the execution of the image processing operations on the PIPE hardware. There is a cycle where no images are input to ensure that the two frames that are subtracted are taken on the same fields. This mapping of the algorithm provides a throughput rate of 6 time cycles, or 100.2 ms, as well as an update rate of 100.2 ms. For our implementation it was crucial to maximize the update rate, not the throughput rate and to maintain it at a constant rate. Therefore, we chose the method outlined in figure 7 which operates on two sets of difference images in parallel. The implementation of this algorithm has the same throughput time of 100.2 ms but a shorter update rate of 66.8 ms.

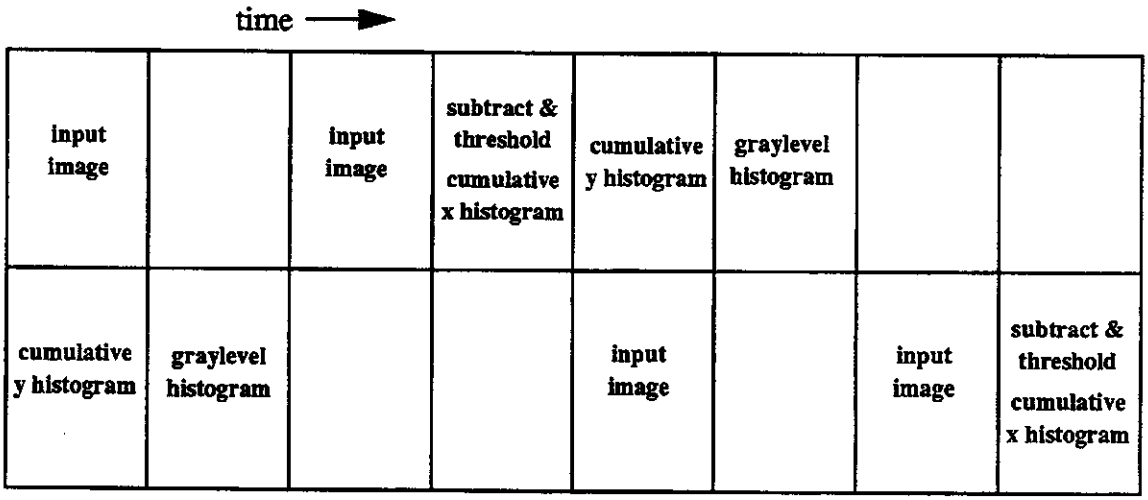


Figure 7. PIPE Implementation Optimized for Update Rate

The remaining NASREM perception processes run on a 68020 board. This includes the Level 1 and Level 2 EX modules, the PIPE manager, and the Level 2 world model process. The execution

time for each process is indicated in the table below:

Process	Execution Time
Level 2 EX	0.7 msec.
Level 1 EX	0.7 msec.
PIPE Manager	63.0 msec.
Level 2 SP	0.5 msec.
Level 2 WM	1.9 msec.

Table 1. Perception Processes' Execution Time

The time noted for the execution of the PIPE manager includes the time that is spent waiting for the proper PIPE cycle in which data is ready, as well as the time to execute the code running on the 68020 system that reads the PIPE histogram values. The resulting update rate for data written by the world model to the Prim level task decomposition process is 66.8 ms, which is the sum of the execution times for all the processes. This implementation shows that the image processing time requirements limit the update rate of three-dimensional information to the manipulator task decomposition system.

7. Conclusions

This paper has described a real-time motion-detection algorithm used to determine three-dimensional position. The algorithm is implemented within the perception hierarchy of the NASREM control system. We have shown the mapping of the algorithm into task decomposition (TD), world modeling (WM) and sensory processing (SP) modules. We have also discussed the modules' implementation in hardware, including the use of special purpose image processing hardware. The system provides real time position data quickly enough for a manipulator to track and catch a ball randomly rolling down an inclined board.

8. References

[1] Albus, J. S., McCain, H. G., Lumia, R.,"NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM)", NIST Technical Note 1235, Gaithersburg, MD, July, 1987.

[2] Aspex, Inc., "PIPE--An Introduction to the PIPE System", New York, 1987.

[3] Chaconas, K. and M. Nashman, "Visual Perception Processing in a Hierarchical Control System", NIST Technical Note 1260, Gaithersburg, MD, March, 1989.

[4] Craig, J.J., "Introduction to Robotics: Mechanics and Control", Addison Wesley Publishing, Massachusetts, 1986.

[5] Fiala, J., "Manipulator Servo Level Task Decomposition", NIST Technical Note 1255, Gaithersburg, MD, October, 1988.

[6] Fiala, J., "Note on NASREM Implementation," NIST Internal Report 89-4215,

Gaithersburg, MD, December, 1989.

[7] Fiala, J. and Wavering, A., "Implementation of a Jacobian Transpose Algorithm," NIST Internal Report 90-4286, Gaithersburg, MD, April, 1990.

[8] Kelmar, L., "Manipulator Primitive Level World Modeling," NIST Technical Note 1273, Gaithersburg, MD, October, 1989.

[9] Kelmar, L. and Lumia, R., "World Modeling for Sensory Interactive Trajectory Generation," to be published at the Third International Symposium on Robotics and Manufacturing (ISRAM), Vancouver, B.C., July, 1990.

[10] Khatib, O., "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation," IEEE Journal of Robotics and Automation, Vol. RA-3, No. 1:43-53, February, 1987.

[11] Wavering, A., "Manipulator Primitive Level Task Decomposition", NIST Technical Note 1256, Gaithersburg, MD, October, 1988.

[12] Yeh, P.S., Barash, S., Wysocki, E., "A Vision System for Safe Robot Operation," Proceedings of the 1988 IEEE Int'l Conference on Robotics and Automation, pp. 1461-1465, April 24-29, 1988.